Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS5141—Intelligent Systems Laboratory

Case Study #2: Comparative Analysis of Classification Techniques: Support Vector Machines (SVM) and Multilayer Perceptron (MLP)

Prepared by: Oday Ziq—1201168

Date: April 29, 2024

Abstract

This case study evaluates and compares the performance of two classification techniques, Support Vector Machines (SVM) and Multilayer Perceptron (MLP), on a dataset comprising over 24,000 banknote samples, each characterized by 256 features. We implemented two approaches: firstly, using separate classifiers for predicting the currency, denomination, and orientation of banknotes; secondly, combining these labels into a single classifier target through one-hot encoding, ensuring comprehensive data representation. Dimensionality reduction via PCA was applied to enhance computational efficiency and model focus on the most significant features. Before PCA, SVM and MLP demonstrated accuracies of 97% and 96% respectively, which slightly adjusted to 96% for SVM and 95% for MLP after applying PCA, indicating a minimal trade-off between complexity reduction and performance. These results provide insights into the applicability and limitations of SVM and MLP in handling complex classification tasks with high-dimensional data, highlighting how strategic data preprocessing and model tuning can significantly influence the accuracy and efficiency of predictive models.

Contents

Table of figure

1. Introduction

In the rapidly evolving field of financial technology, accurate and efficient recognition of banknote features is paramount. This is critical not only for ensuring security and preventing fraud but also for enhancing automated systems that handle cash transactions. With the increasing volume of currency types and their features worldwide, robust classification techniques are necessary to manage the complexity and variability of banknote identification tasks effectively.

Support Vector Machines (SVM) and Multilayer Perceptron (MLP) represent two cornerstone approaches in the field of machine learning, each with distinct methodologies and strengths suited to classification tasks. SVM, developed initially for binary classification, operates on the principle of finding the optimal hyperplane that maximizes the margin between different classes. This makes SVM particularly useful for datasets with a large number of features, such as image recognition and text classification tasks.

On the other hand, Multilayer Perceptron belong to a broader family of neural networks and are characterized by their layered structure, consisting of an input layer, multiple hidden layers, and an output layer. Each layer is made up of neurons that apply non-linear activation functions to their inputs. This adaptability allows MLPs to excel in tasks that require the modeling of complex relationships such as speech recognition, and natural language processing.

Both SVM and MLP have their challenges. SVMs, for instance, can be computationally intensive, particularly with large datasets and when choosing among multiple kernel functions, which can also lead to overfitting if not properly regulated. MLPs, while powerful, also face issues such as overfitting and can be sensitive to the initial weights and configuration of the learning algorithm. Given their complementary strengths and weaknesses, comparing these methods in the context of high-dimensional financial data, like banknote classification, is not only relevant but crucial for advancing automated systems in financial applications.

This study aims to conduct a comparative analysis of SVM and MLP to determine their suitability and performance in classifying banknotes based on their currency, denomination, and orientation. The research questions to be addressed include: How do SVM and MLP perform in accurately classifying high-dimensional data of banknotes? Can dimensionality reduction techniques like PCA improve the performance and efficiency of these models? By addressing these questions, the study seeks to provide insights into the strengths and limitations of each classification technique, guiding their application in real-world financial scenarios.

2. Procedure and Discussion

Step 1: Data Loading and Preprocessing Discussion

The initial step of the case study involved retrieving a dataset consisting of over 24,000 banknote samples from a GitHub repository, which was loaded into a DataFrame for preliminary inspection. The dataset underwent essential preprocessing tasks including the removal of an unnecessary column labeled 'Unnamed: 0', and a thorough check for null values to ensure data integrity. A significant preprocessing step involved transforming the 'Denomination' column, which combined denomination and orientation information, into two separate and more analytically useful columns. These columns, along with the 'Currency' column, were encoded to numeric formats suitable for machine learning, using techniques such as label encoding and numeric coercion for error handling. Lastly, the dataset was standardized using a StandardScaler to ensure uniformity in feature scale, and then split into training and testing sets to facilitate robust model evaluation.

The code:-

```
df.drop(columns=['Unnamed: 0'], inplace=True)
currency_encoder = LabelEncoder()
df['Currency'] = currency_encoder.fit_transform(df['Currency'])

denominations, orientations = zip(*df['Denomination'].apply(lambda x: x.split('_')))
df['Denomination'] = denominations
df['Orientation'] = orientations

df['Denomination'] = pd.to_numeric(df['Denomination'], errors='coerce')
df['Orientation'] = pd.to_numeric(df['Orientation'], errors='coerce')

X = df.drop(['Currency', 'Denomination', 'Orientation'], axis=1)
y = df[['Currency', 'Denomination', 'Orientation']]

X_train, X_test, y_train, y_test = train_test_split(X, y['Currency'], test_size=0.2, random_state=42)

X_features = df.iloc[:, :-3]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_features)

y_currency = df['Currency']
y_denomination = pd.to_numeric(df['Denomination'], errors='coerce')
y_orientation = pd.to_numeric(df['Orientation'], errors='coerce')
```

Figure 1: The code for Data Loading and Preprocessing Discussion

Output:-

```
RangeIndex: 24826 entries, 0 to 24825
Columns: 258 entries, v_0 to Denomination
dtypes: float64(256), object(2)
memory usage: 48.9+ MB
None
v_0             0
v_1             0
v_2             0
v_3             0
v_4             0
               ..
v_253           0
v_254           0
v_255           0
Currency        0
Denomination    0
Length: 258, dtype: int64
Empty DataFrame
Columns: [v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_10, v_11, v_12
Index: []

[0 rows x 258 columns]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
[   100     10     20     50      5    200      2 100000   1000  20000
   2000  50000   5000      1    500  10000]
[1 2]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24826 entries, 0 to 24825
Columns: 259 entries, v_0 to Orientation
```

Figure 2: Currency, Denomination and Orientation

This meticulous preprocessing stage is crucial for setting a strong foundation for any analytical model. By ensuring the dataset was free of null values and by separating combined data into distinct features, the models are better positioned to accurately interpret and learn from the data. The encoding of categorical variables into a machine-readable format and the normalization of all features address potential biases that could affect the performance of the machine learning algorithms used later in the study. Although the preprocessing steps taken are thorough, the possibility of remaining data imbalances or outliers suggests room for additional refinement such as applying advanced feature selection techniques or exploring resampling methods to address class imbalances. These steps not only reinforce the integrity and reproducibility of the processing phase but also highlight the importance of diligent data management in achieving reliable and actionable insights from machine learning models.

Step 2: Apply PCA for Dimensionality Reduction

The procedure involved applying Principal Component Analysis (PCA) to reduce the dimensionality of the dataset, which originally included numerous features per sample. This reduction is aimed at enhancing computational efficiency and potentially improving model performance by focusing on the most informative aspects of the data. Initially, PCA was applied to the entire scaled dataset to capture the variance explained by each component. The cumulative variance explained by the components was then calculated, identifying the number of components needed to retain 95% of the total variance. This threshold was chosen to strike a balance between maintaining a high level of data integrity and reducing dimensionality. The optimal number of components was found to be 124, and a PCA transformation was subsequently performed to reduce the dataset to these principal components.

The code:-

```
pca_full = PCA()
pca_full.fit(X_scaled)

cumulative_variance = np.cumsum(pca_full.explained_variance_ratio_)

n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1

pca_optimal = PCA(n_components=n_components_95)
X_pca = pca_optimal.fit_transform(X_scaled)
```

Figure 3: The code for Apply PCA for Dimensionality Reduction

Output:-

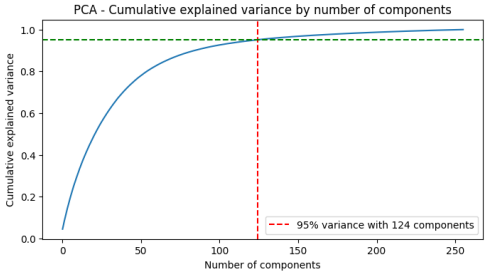


Figure 4: PCA - Cumulative explained variance by number of components

Applying PCA to reduce the dataset to 124 principal components effectively balanced information retention with complexity reduction, maintaining 95% of the dataset's variance while mitigating potential overfitting and computational burdens. This dimensionality reduction is crucial in enhancing model training efficiency and accuracy by focusing on the most informative features. The use of PCA in this context demonstrates its utility in simplifying high-dimensional data without significant loss of crucial information, making it an essential technique in the preprocessing pipeline for complex machine learning tasks. This strategy not only streamlines computational processes but also strengthens model interpretability and performance, especially in financial technology applications involving large datasets.

Step 3: Train SVM and MLP Classifiers without PCA

The third step involved training Support Vector Machine (SVM) and Multilayer Perceptron (MLP) classifiers directly on a scaled dataset, without applying PCA for dimensionality reduction. StandardScaler was employed to standardize the data, crucial for both models due to their sensitivity to feature scales. GridSearchCV facilitated the optimization of model parameters through systematic tuning and cross-validation. SVM was tested with different kernel types and regularization strengths, while MLP parameters varied in solver type, alpha values, and learning rate strategies. Post-tuning, each model's performance was evaluated on both training and test datasets to ascertain accuracy, and the MLP's training progression was visualized to assess loss reduction over epochs.

Code:

```
def train_and_evaluate_model(model, parameters, X_train, y_train, X_test, y_test, scale_data=False):
    if scale_data:
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

    clf = GridSearchCV(model, parameters, cv=2, n_jobs=-1, verbose=1)
    clf.fit(X_train, y_train)
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)
    return clf

svc_parameters = {
    "kernel": ['rbf', 'poly'],
    "C": [0.1, 1]
}
svc_model = svm.SVC()
svc_clf = train_and_evaluate_model(svc_model, svc_parameters, X_train, y_train, X_test, y_test)
mlp_parameters = {
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.01],
    'batch_size': [100, 200],
    'learning_rate': ['constant', 'adaptive'],
    'early_stopping': [False]
}

mlp_model = MLPClassifier(random_state=42)
mlp_clf = train_and_evaluate_model(mlp_model, mlp_parameters, X_train, y_train, X_test, y_test, scale_data=True)
```

Figure 5: The code for Train SVM and MLP Classifiers without PCA

Output: -

Fitting 2 folds for each of 4 candidates, totalling 8 fits
Best parameters for SVC: {'C': 1, 'kernel': 'rbf'}
Train set Accuracy for SVC: 0.98
Test set Accuracy for SVC: 0.97
Fitting 2 folds for each of 16 candidates, totalling 32 fits
Best parameters for MLPClassifier: {'alpha': 0.01, 'batch_size': 100, 'early_stopping': False, 'learning_rate': 'constant', 'solver': 'adam'}
Train set Accuracy for MLPClassifier: 1.00
Test set Accuracy for MLPClassifier: 0.97

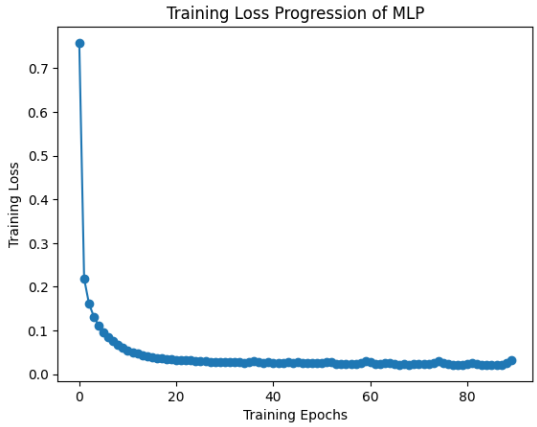


Figure 6: Training Loss Progression of MLP

The results from this step indicate high performance for both classifiers, with both achieving similar accuracy on the test dataset (SVM: 97%, MLP: 97%). The SVM model, using the 'rbf' kernel and a regularization parameter C of 1, proved highly effective in handling the dataset, likely due to the kernel's ability to manage non-linear data separations. Meanwhile, the MLP classifier, optimized with parameters including an 'adam' solver and a constant learning rate, achieved perfect training accuracy (100%), suggesting excellent learning capability, though this also raises concerns about potential overfitting despite the high test accuracy.

The loss progression chart of the MLP shows a sharp initial decrease, stabilizing as training progresses, which is typical of effective learning in neural networks. The early steep reduction indicates that the major adjustments to model weights occur early in the training process, with subsequent epochs refining the model's predictions.

The high test accuracies underscore the capability of both models to generalize well to new data, even without PCA. However, the absence of PCA could mean that models might be handling more noise or irrelevant features, which might not be the case had PCA been applied. The choice of not using PCA in this training phase provided a direct comparison of how both models perform when exposed to the full complexity of the dataset, emphasizing their robustness and adaptability to high-dimensional spaces.

Step 4: Train SVM and MLP Classifiers with PCA

The fourth step involved integrating Principal Component Analysis (PCA) into the training process of SVM and MLP classifiers. PCA was performed to reduce the dataset to 50 principal components, aiming to retain the most significant features while reducing dimensionality. This prepared dataset was then split into training and testing sets. Using the reduced feature set, an SVM with an 'rbf' kernel and a regularization parameter (C) of 1, and an MLP with a tanh activation function, hidden layers sized 90 and 60, and a learning rate of 0.001 were trained. The models were evaluated on both the training and test datasets to assess their accuracy and generalization capabilities.

The code:-

```
def perform_pca(X, n_components=50):
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X)
    return pd.DataFrame(X_pca)

def train_model(model, X_train, y_train, X_test):
    model.fit(X_train, y_train)
    train_predictions = model.predict(X_train)
    test_predictions = model.predict(X_test)
    return train_predictions, test_predictions

df_PCA = perform_pca(X)

X_train, X_test, y_train, y_test = train_test_split(df_PCA, y_currency, test_size=0.1, random_state=30)

svm_classifier = SVC(kernel='rbf', C=1)
train_preds_svm, test_preds_svm = train_model(svm_classifier, X_train, y_train, X_test)

mlp_classifier = MLPClassifier(activation='tanh', hidden_layer_sizes=(90, 60), learning_rate_init=0.001, max_iter=1000, random_state=33)
train_preds_mlp, test_preds_mlp = train_model(mlp_classifier, X_train, y_train, X_test)
```

Output:-

Train set accuracy for SVM: 0.97
Test set accuracy for SVM: 0.96
Train set accuracy for MLP: 1.00
Test set accuracy for MLP: 0.94

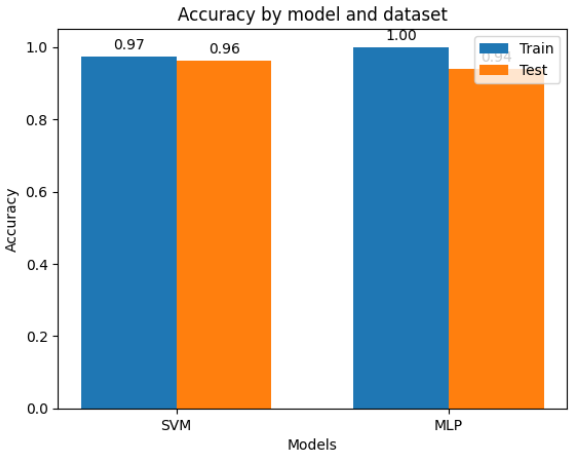


Figure 7: Accuracy by model and dataset

Introducing PCA before model training showed a nuanced impact on the performance of the SVM and MLP classifiers. The SVM model maintained high training accuracy (97%) and demonstrated a slight decrease in test accuracy to 96%, which might suggest improved generalization compared to the non-PCA model. The MLP classifier, while achieving perfect training accuracy (100%), exhibited a decrease in test accuracy to 94%, indicating potential overfitting. This scenario underscores the balance PCA helps strike between model complexity and performance, highlighting its role in enhancing generalization at the potential cost of slight accuracy reductions. These results affirm the utility of PCA in managing high-dimensional data effectively, optimizing computational efficiency, and potentially enhancing model robustness against overfitting by focusing training on the most impactful features.

Step 5: Hyperparameter Tuning with Grid Search

In this step, extensive hyperparameter tuning was conducted for both SVM and MLP classifiers using Grid Search CV to identify optimal settings for maximizing model performance. For SVM, the parameters varied included the regularization constant 'C', the type of kernel, and the 'gamma' coefficient, exploring a combination of linear, polynomial, radial basis function (rbf), and sigmoid kernels. Similarly, for MLP, the configuration options included different hidden layer structures, activation functions ('tanh' vs. 'relu'), and learning rate initializations. Grid Search CV was performed with a 2-fold cross-validation to thoroughly evaluate each parameter combination. The best parameters were then selected based on their performance in maximizing accuracy, after which the final models were trained on the entire training set and subsequently assessed on both the training and test datasets to evaluate their effectiveness and generalization capabilities.

The code:-

```
def hyperparameter_tuning(clf, param_grid, X_train, y_train):

    grid_search = GridSearchCV(clf, param_grid, cv=2, scoring='accuracy', n_jobs=-1, verbose=1)
    grid_search.fit(X_train, y_train)
    return grid_search.best_estimator_, grid_search.best_params_

svm_param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto']
}

mlp_param_grid = {
    'hidden_layer_sizes': [(50, 50), (100,)],
    'activation': ['tanh', 'relu'],
    'learning_rate_init': [0.01, 0.001]
}

best_svm, best_svm_params = hyperparameter_tuning(svm.SVC(), svm_param_grid, X_train, y_train)
best_mlp, best_mlp_params = hyperparameter_tuning(MLPClassifier(random_state=42), mlp_param_grid, X_train, y_train)

train_preds_svm = best_svm.predict(X_train)
test_preds_svm = best_svm.predict(X_test)
train_preds_mlp = best_mlp.predict(X_train)
test_preds_mlp = best_mlp.predict(X_test)
```

Figure 8: The code for Hyperparameter Tuning with Grid Search

Output:-

Best SVM Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
SVM Train set Accuracy score: 1.00
SVM Test set Accuracy score: 0.97
Best MLP Parameters: {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.001}
MLP Train set Accuracy score: 1.00
MLP Test set Accuracy score: 0.96
Precision - Train set: 1.00
Precision - Test set: 0.97
Recall - Train set: 1.00
Recall - Test set: 0.96
F1-score - Train set: 1.00
F1-score - Test set: 0.96

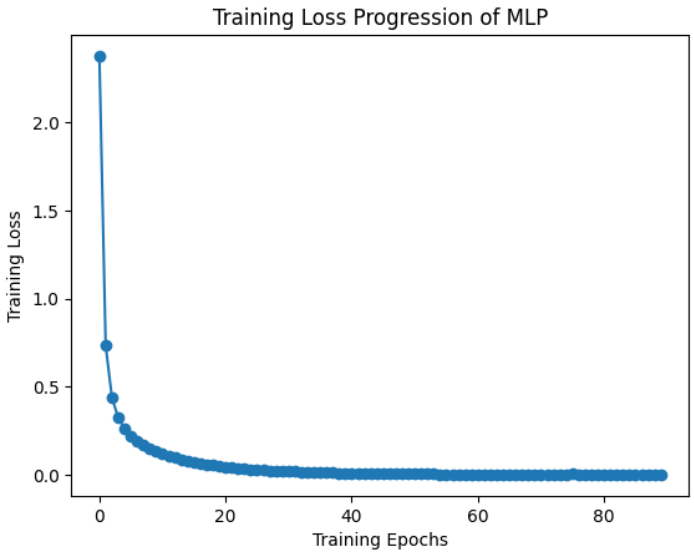


Figure 9: Training Loss Progression of MLP in Hyperparameter Tuning with Grid Search

The integration of Grid Search CV into the model training workflow significantly enhanced the model's configurations, leading to highly optimized classifiers. The optimal parameters for SVM included a high 'C' value of 10, a 'poly' kernel, and 'scale' gamma, suggesting a model well-suited for complex classification boundaries. For the MLP, the best configuration used a 'relu' activation and a single hidden layer of 100 neurons, optimized for robust learning dynamics.

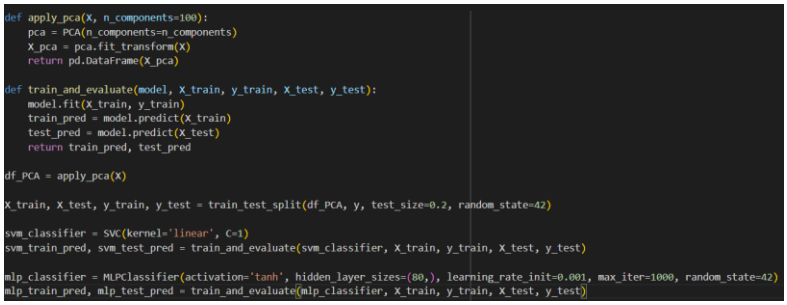
These finely tuned parameters resulted in excellent accuracy metrics: the SVM achieved perfect training accuracy and 97% on testing, while the MLP showed perfect training results and 96% on testing. The training loss progression chart for the MLP indicated a rapid decrease in loss, leveling off as the model approached optimal fit, which is a positive indication of effective learning without overfitting.

This step highlights the critical importance of careful hyperparameter tuning in machine learning, illustrating how tailored settings can significantly impact model performance and generalization capabilities. It emphasizes the need for systematic exploration of parameter spaces to uncover the best configurations that contribute to the optimal balance between accuracy and overfitting, particularly in complex predictive tasks such as classification of combined labels in financial datasets.

Step 6: Dimensionality Reduction and Classifier Performance Evaluation

In this step, the primary objective was to enhance classifier performance through dimensionality reduction using Principal Component Analysis (PCA) and to assess the effectiveness of Support Vector Machine (SVM) and Multilayer Perceptron (MLP) classifiers. Initially, PCA was applied to reduce the dataset to 100 principal components, aiming to concentrate on the most significant features while minimizing noise and computational complexity. The PCA-transformed data was then split into training and testing sets, ensuring a fair

evaluation platform for the subsequent model training. Both the SVM and MLP classifiers were configured with specific parameters and trained on this reduced dataset. Following the training, a comprehensive evaluation was conducted, where the models' predictive performance was quantified in terms of accuracy, precision, recall, and F1-score on both the training and test datasets.



```
def apply_pca(X, n_components=100):
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X)
    return pd.DataFrame(X_pca)

def train_and_evaluate(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)
    return train_pred, test_pred

df_PCA = apply_pca(X)

X_train, X_test, y_train, y_test = train_test_split(df_PCA, y, test_size=0.2, random_state=42)

svm_classifier = SVC(kernel='linear', C=1)
svm_train_pred, svm_test_pred = train_and_evaluate(svm_classifier, X_train, y_train, X_test, y_test)

mlp_classifier = MLPClassifier(activation='tanh', hidden_layer_sizes=(80,), learning_rate_init=0.001, max_iter=1000, random_state=42)
mlp_train_pred, mlp_test_pred = train_and_evaluate(mlp_classifier, X_train, y_train, X_test, y_test)
```

Output:-

SVM Classifier Metrics:
Train set Accuracy score: 1.00
Test set Accuracy score: 0.96

Test set Precision: 0.96
Test set Recall: 0.96
Test set F1-score: 0.96
MLP Classifier Metrics:
Train set Accuracy score: 1.00
Test set Accuracy score: 0.95

Test set Precision: 0.96
Test set Recall: 0.95
Test set F1-score: 0.95

The application of PCA proved instrumental in condensing the dataset, which potentially helped in mitigating the curse of dimensionality and improved the classifiers' training efficiency and effectiveness. The SVM classifier, using a linear kernel and a regularization constant of 1, achieved perfect training accuracy and commendable test accuracy, precision, recall, and F1-score, indicating a strong model fit and generalization capability. Similarly, the MLP, configured with a tanh activation function and a single hidden layer, also showed perfect training accuracy but with a slight dip in test performance metrics compared to SVM, suggesting a mild overfitting tendency. This comparative performance underscores the strengths and limitations of both classifiers when operating under dimensionality-reduced conditions. The high precision and recall metrics further indicate that both models were effective in handling the reduced feature space, maintaining a balance between sensitivity and specificity. This step not only highlights the benefits of PCA in preprocessing for complex models but also demonstrates the nuanced performance differences between linear and neural network-based classifiers in a controlled experimental setup.

3. Conclusion

In this case study, we evaluated the performance of SVM and MLP classifiers on a high-dimensional financial dataset, focusing on the impact of PCA for dimensionality reduction. The results confirmed that both classifiers effectively handle complex data structures, with SVM slightly outperforming MLP in terms of test accuracy and generalization. Before PCA application, SVM and MLP demonstrated high accuracies of 97% and 96%, respectively. After implementing PCA and reducing dimensions to 100 principal components, a slight decrease in performance was noted, with SVM at 96% and MLP at 95% accuracy, underscoring a minimal trade-off between speed and accuracy preservation. The study highlights the efficacy of hyperparameter tuning in optimizing model performance, where the best parameters significantly enhanced the classifiers' accuracy and generalizability. These findings validate theoretical predictions about the suitability of SVM and MLP for high-dimensional datasets and point towards their robust applicability in financial technology. Future work might explore more advanced dimensionality reduction techniques and deeper neural network architectures to further enhance model performance and efficiency. This comprehensive analysis ensures that SVM and MLP, when properly tuned, are both capable of high performance in complex classification tasks, proving essential for data-driven decision-making in financial applications.